IIR Filter Design Midsem Assignment

Name : Prajwal Nayak \ Roll Number : 22B4246 \

Reviewer: \ Name : Sarvadnya Purkar \ Roll Number : 22B4232 \

Given Specifications:

1. Filter Number (M): 114
2. Given that sampling frequency f_s = 630kHz
3. We have to design FILTER TYPE I : where passband and stopband are both monotonic => we use butterworth filter.
4. Tolerances : $\delta_1 = \delta_2 = 0.15$
5. Transition Band : 5kHz on either side of each passband.
6. To find ranges of passband, M = 11Q + R. Q = M//11 and R = M % 10. D for Group I bands is Q and for Group II is R.
7. Group I of Frequency Bands: The frequency band in this group is $(40 + 5D)\ to\ (70 + 5D)$
8. Group II of Frequency Bands: The frequency band in this group is $(170 + 5D)\ to\ (200 + 5D)$

```
import math  # Importing the math module for mathematical operations
import pandas as pd  # Bringing in pandas for structured data
representation
import matplotlib.pyplot as plt  # Importing the plotting library
from numpy.polynomial import Polynomial
from sympy import symbols, Poly, I, Abs, arg, lambdify, simplify, CC,
tan
import numpy as np

M = 114
Q = M//11
R = M%10
print("The value of Q is: ",Q)
print("The value of R is: ",R)
```

```
The value of Q is:  10
The value of R is:  4
```

```
print("Group I frequency band range is from",40+5*(Q),"to",70+5*(Q))
print("Group II frequency band range is
from",170+5*(R),"to",200+5*(R))
```

```
Group I frequency band range is from 90 to 120
Group II frequency band range is from 190 to 220
```

```
print("Transition band for Group I frequency band is",40+5*(Q)-
5,"to",40+5*(Q)," and ",70+5*(Q),"to",70+5*(Q)+5)
```

```
print("Transition band for Group II frequency band is",170+5*(R)-
5,"to",170+5*(R)," and ",200+5*(R),"to",200+5*(R)+5)
```

```
Transition band for Group I frequency band is 85 to 90  and  120 to
125
Transition band for Group II frequency band is 185 to 190  and  220 to
225
```

##Bilinear Transformation $s=\frac{1-z^{-1}}{1+z^{-1}}$ \ For $z=e^{j\omega}$, we get $s=jtan(\omega/2)$ \ For $s=j\Omega$, we get $\Omega=tan(\omega/2)$

Considering that the filter decreases once it enters the stopband, we won't change the tolerances initially. In the end, we will check if the tolerances are being violated and if that happens we will reduce our tolerances so that our result is accurate and fulfils the required tolerance specifications.

```
import math
f = [0,85,90,120,125,185,190,220,225,315]
w = []
Omega = []
for i in f:
  print("for f = ",i,", we have w = ",(2*math.pi*i)/630,"and Analog
Angular frequency is ",(math.tan((2*math.pi*i)/(2*630))))
  w.append((2*math.pi*i)/630)
  Omega.append(math.tan((2*math.pi*i)/(2*630)))
w[9] = math.pi
Omega[9] = math.inf
print(w)
print(Omega)
```

```
for f =  0 , we have w =  0.0 and Analog Angular frequency is  0.0
for f =  85 , we have w =  0.8477313509686744 and Analog Angular
frequency is  0.4512171831783031
for f =  90 , we have w =  0.8975979010256552 and Analog Angular
frequency is  0.4815746188075286
for f =  120 , we have w =  1.19679720136754 and Analog Angular
frequency is  0.6817884558007686
for f =  125 , we have w =  1.246663751424521 and Analog Angular
frequency is  0.7189510382878603
for f =  185 , we have w =  1.8450623521082912 and Analog Angular
frequency is  1.3201833136548846
for f =  190 , we have w =  1.8949289021652722 and Analog Angular
frequency is  1.3909153012442141
for f =  220 , we have w =  2.1941282025071573 and Analog Angular
frequency is  1.9505721632508881
for f =  225 , we have w =  2.243994752564138 and Analog Angular
frequency is  2.0765213965723364
for f =  315 , we have w =  3.141592653589793 and Analog Angular
```

```
frequency is  1.633123935319537e+16
[0.0, 0.8477313509686744, 0.8975979010256552, 1.19679720136754,
1.246663751424521, 1.8450623521082912, 1.8949289021652722,
2.1941282025071573, 2.243994752564138, 3.141592653589793]
[0.0, 0.4512171831783031, 0.4815746188075286, 0.6817884558007686,
0.7189510382878603, 1.3201833136548846, 1.3909153012442141,
1.9505721632508881, 2.0765213965723364, inf]
```

```
import pandas as pd
data = {
    "Category": ["Stopband for Group 1", "Passband for Group 1",
"Stopband intermediate", "Passband for Group 2","Stopband for Group
2"],
    "Type":
["Monotonic","Monotonic","Monotonic","Monotonic","Monotonic"],
    "Un-normalised Frequency(kHz)[f]": ["0-85", "90-120", "125-
185","190-220","225-315"],
    "Normalised Frequency(rad/s)[ω]": [f"{w[0]}-{w[1]}", f"{w[2]}-
{w[3]}", f"{w[4]}-{w[5]}",f"{w[6]}-{w[7]}",f"{w[8]}-{w[9]}"],
    "Analog Angular Frequency(rad/s)[Ω]":[f"{Omega[0]}-
{Omega[1]}",f"{Omega[2]}-{Omega[3]}",f"{Omega[4]}-
{Omega[5]}",f"{Omega[6]}-{Omega[7]}",f"{Omega[8]}-{Omega[9]}"]
}
df = pd.DataFrame(data)
df
```

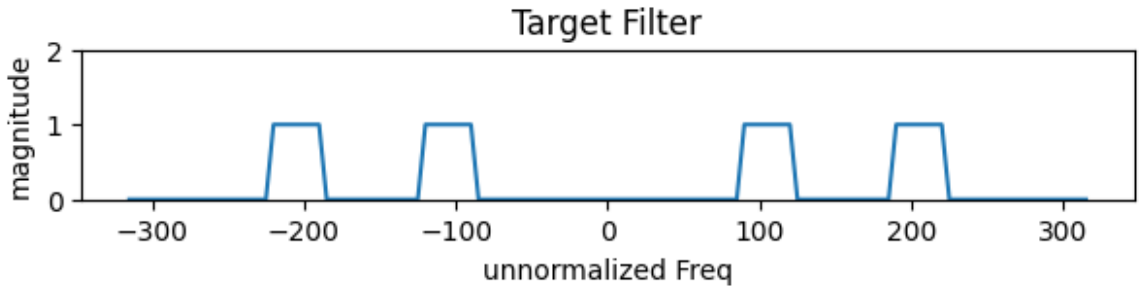
	Category	Type	Un-normalised Frequency(kHz)[f]
0	Stopband for Group 1	Monotonic	0-85
1	Passband for Group 1	Monotonic	90-120
2	Stopband intermediate	Monotonic	125-185
3	Passband for Group 2	Monotonic	190-220
4	Stopband for Group 2	Monotonic	225-315

	Normalised Frequency(rad/s)[ω]
0	0.0-0.8477313509686744
1	0.8975979010256552-1.19679720136754
2	1.246663751424521-1.8450623521082912
3	1.8949289021652722-2.1941282025071573
4	2.243994752564138-3.141592653589793

	Analog Angular Frequency(rad/s)[Ω]
0	0.0-0.4512171831783031
1	0.4815746188075286-0.6817884558007686
2	0.7189510382878603-1.3201833136548846
3	1.3909153012442141-1.9505721632508881
4	2.0765213965723364-inf

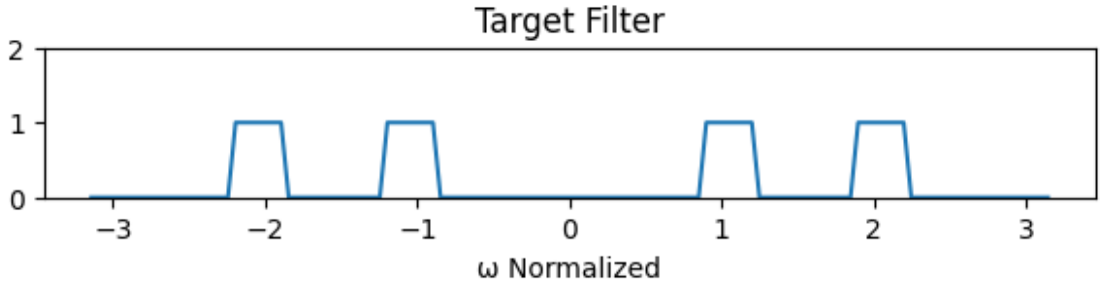
##What is our Target?

```
import matplotlib.pyplot as plt
val = [0,0,1,1,0,0,1,1,0,0]
f2 = f[::-1]
val2 = val[::-1]
Omega2 = Omega[::-1]
w2 = w[::-1]
f3=[i*-1 for i in f2]+f
w3=[i*-1 for i in w2]+w
Omega3 = [i*-1 for i in Omega2]+Omega
valNeg = val2+val
plt.figure(figsize=(7,1))
plt.plot(f3, valNeg)
plt.title('Target Filter')
plt.ylim(0,2)
plt.xlabel('unnormalized Freq')
plt.ylabel('magnitude')
plt.show()
```



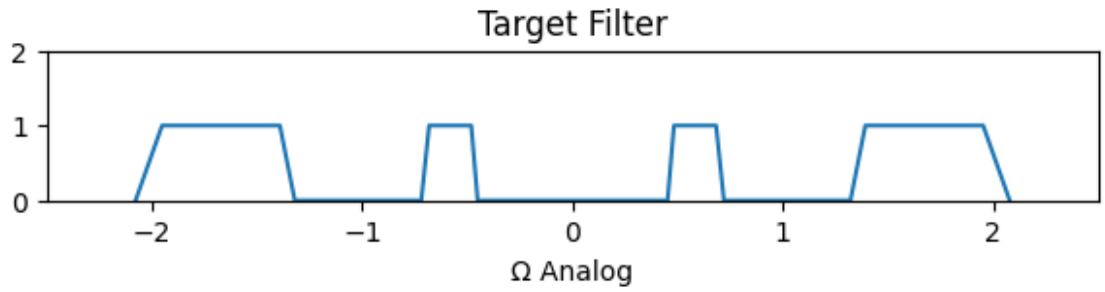
The above is the unnormalized filter which we are going to design

```
plt.figure(figsize=(7,1))
plt.plot(w3, valNeg)
plt.ylim(0,2)
plt.xlabel('ω Normalized')
plt.title("Target Filter")
plt.show()
```




```
plt.figure(figsize=(7,1))
plt.plot(Omega3, valNeg)
```

```
plt.ylim(0,2)
plt.xlim(-2.5,2.5)
plt.xlabel('Ω Analog')
plt.title("Target Filter")
plt.show()
```



The approach I will be using is P3 : using two Bandpass filters in parallel.

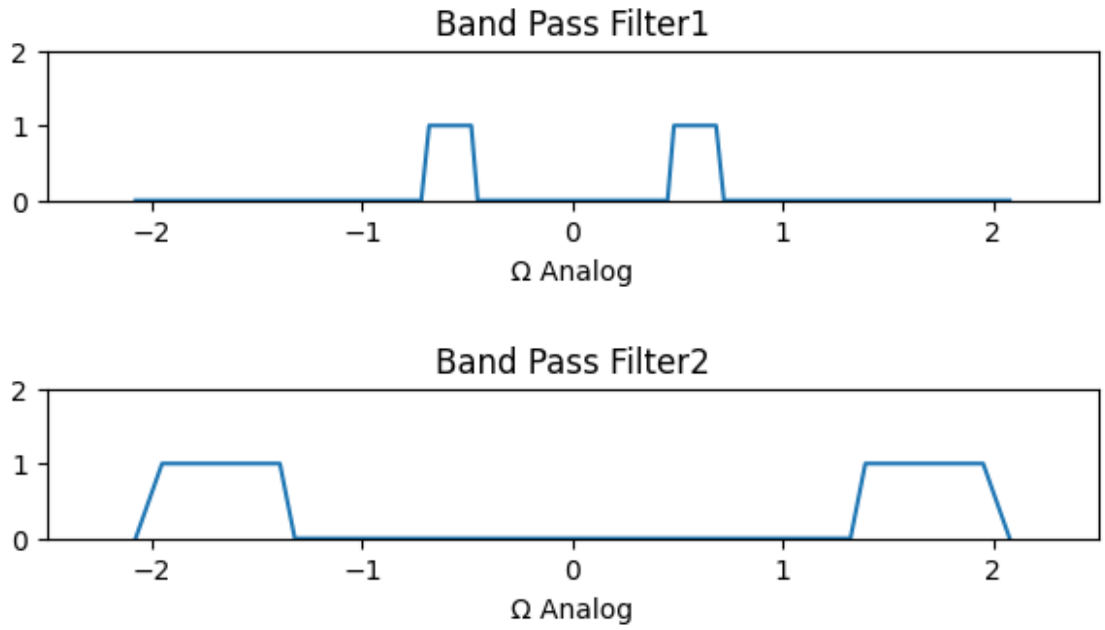
##Parallel BandPass Filters

```
print(f3)
print(w3)
print(Omega3)
```

```
[-315, -225, -220, -190, -185, -125, -120, -90, -85, 0, 0, 85, 90,
120, 125, 185, 190, 220, 225, 315]
[-3.141592653589793, -2.243994752564138, -2.1941282025071573, -
1.8949289021652722, -1.8450623521082912, -1.246663751424521, -
1.19679720136754, -0.8975979010256552, -0.8477313509686744, -0.0, 0.0,
0.8477313509686744, 0.8975979010256552, 1.19679720136754,
1.246663751424521, 1.8450623521082912, 1.8949289021652722,
2.1941282025071573, 2.243994752564138, 3.141592653589793]
[-inf, -2.0765213965723364, -1.9505721632508881, -1.3909153012442141,
-1.3201833136548846, -0.7189510382878603, -0.6817884558007686, -
0.4815746188075286, -0.4512171831783031, -0.0, 0.0,
0.4512171831783031, 0.4815746188075286, 0.6817884558007686,
0.7189510382878603, 1.3201833136548846, 1.3909153012442141,
1.9505721632508881, 2.0765213965723364, inf]
```

```
val_pb1=[0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0]
val_pb2=[0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0]
plt.figure(figsize=(7,1))
plt.plot(Omega3,val_pb1)
plt.ylim(0,2)
plt.xlim(-2.5,2.5)
plt.xlabel('Ω Analog')
plt.title("Band Pass Filter1")
plt.show()
plt.figure(figsize=(7,1))
plt.plot(Omega3,val_pb2)
```

```
plt.ylim(0,2)
plt.xlim(-2.5,2.5)
plt.xlabel('Ω Analog')
plt.title("Band Pass Filter2")
plt.show()
```



$$H_{\text{final,analog}}(\Omega)=H_{\text{BP,analog}}(\Omega)+H_{\text{BS,analog}}(\Omega)$$

$$H_{\text{final}}(e^{j\omega})=H_{\text{BP}}(e^{j\omega})+H_{\text{BS}}(e^{j\omega})$$

$$h_{\text{final}}[n]=h_{\text{BP}}[n]+h_{\text{BS}}[n]$$

##Band Pass Filter 1

```
pb1_p1 = Omega3[12]
pb1_p2 = Omega3[13]
pb1_s1 = Omega3[11]
pb1_s2 = Omega3[14]
pb1_Omega_sq = pb1_p1*pb1_p2
pb1_B = (pb1_p2-pb1_p1)
pb1_ls1 = (pb1_s1**2 - pb1_Omega_sq)/(pb1_B*pb1_s1)
pb1_ls2 = (pb1_s2**2 - pb1_Omega_sq)/(pb1_B*pb1_s2)
print("Omega_sqared = ",pb1_Omega_sq)
print("B = ",pb1_B)
print('Ω LS1=',pb1_ls1)
print('Ω LS2=',pb1_ls2)
```

```
Omega_sqared =  0.3283320157096287
B =  0.20021383699324002
```

```
Ω LS1= -1.3807302950558302
Ω LS2= 1.3099444957834618
```

We choose minimum of the magnitude of the two

```
LP1_stop = min(abs(pb1_ls1),abs(pb1_ls2))
print("stopband edge for Low pass is ",LP1_stop)
LP1_pass = 1
tol = 0.15
D1 = 1/(1-tol)**2 -1
D2 = 1/tol**2 -1
print("The value of D1 is",D1)
print("The value of D2 is",D2)
```

```
stopband edge for Low pass is  1.3099444957834618
The value of D1 is 0.3840830449826991
The value of D2 is 43.44444444444444
```

$$N \geq \frac{\log\left(\frac{D_2}{D_1}\right)}{2\log\left(\frac{\Omega_S}{\Omega_P}\right)}$$

```
import numpy as np
N_min = int(np.ceil(np.log(D2/D1)/(2*np.log(LP1_stop/LP1_pass))))
print('N_min = ',N_min)
```

```
N_min =  9
```

We choose **N** = (N_{\text{min}} = 20)

$$\frac{\Omega_P}{\left(D_1\right)^{\frac{1}{2N}}} \leq \Omega_C \leq \frac{\Omega_S}{\left(D_2\right)^{\frac{1}{2N}}}$$

```
N = N_min
leftTerm = LP1_pass/(D1**(1/(2*N)))
rightTerm = LP1_stop/(D2**(1/(2*N)))
print(leftTerm,'<= Ω c <=',rightTerm)
```

```
1.0545993333022403 <= Ω c <= 1.0623229092896456
```

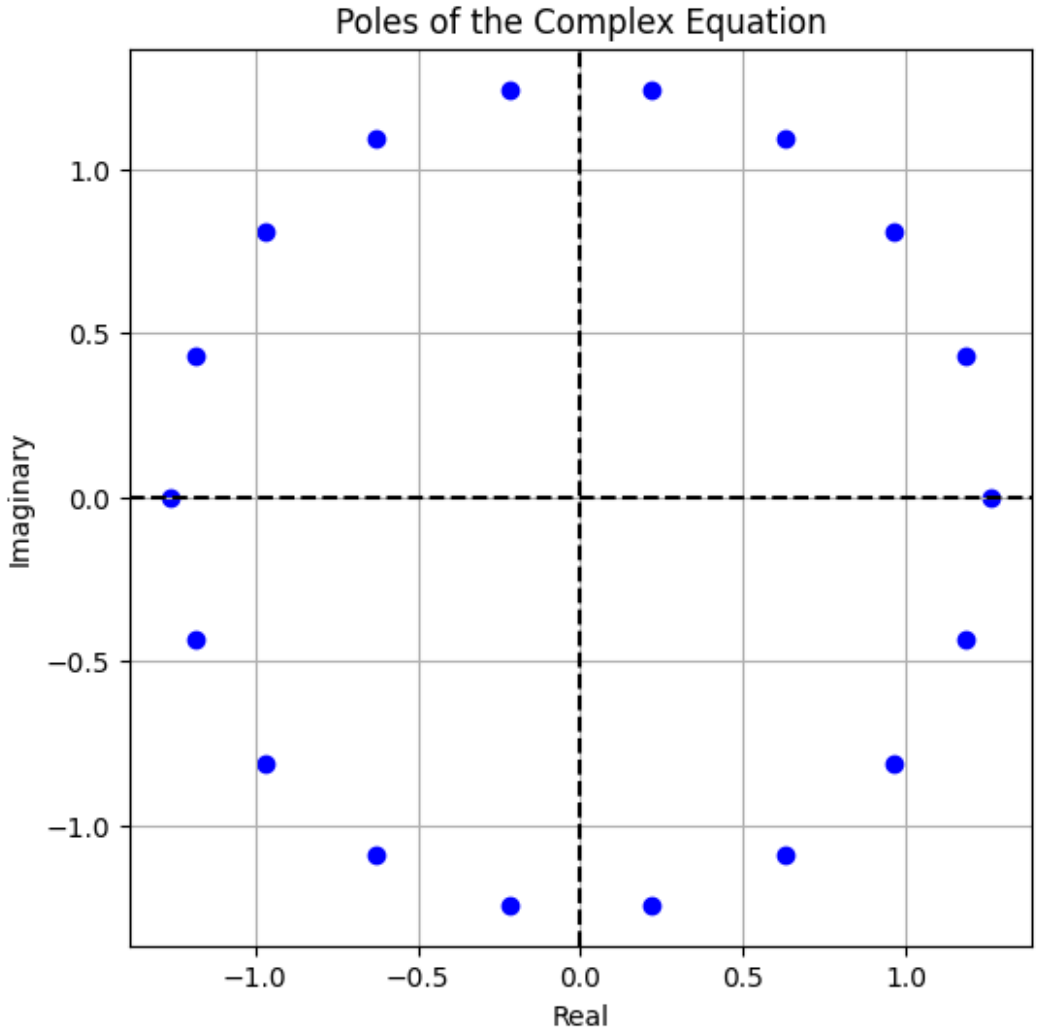
```
lp1_omegac = (leftTerm+rightTerm)/2
print("Omega_c = ",lp1_omegac)
```

```
Omega_c =  1.058461121295943
```

butterworth filter equation:

$$\left|H_{\text{analog, LPF}}(s)\right|^2 = \frac{1}{1+\left(\frac{s}{1.05846\,j}\right)^{2\times 9}}$$

```
poles = [lp1_omegac * np.exp((np.pi + 2 * k * np.pi) / (2 * N)) for k
in range(0, 2*N)*np.exp(1j*np.pi/2)]
plt.figure(figsize=(6, 6))
plt.scatter(np.real(poles), np.imag(poles), color='blue', marker='o')
plt.axhline(y=0, color='k', linestyle='--')
plt.axvline(x=0, color='k', linestyle='--')
plt.ylabel('Imaginary')
plt.xlabel('Real')
plt.title('Poles of the Complex Equation')
plt.grid(True)
plt.show()
stable_poles = [pole for pole in poles if np.real(pole) < 0]
```

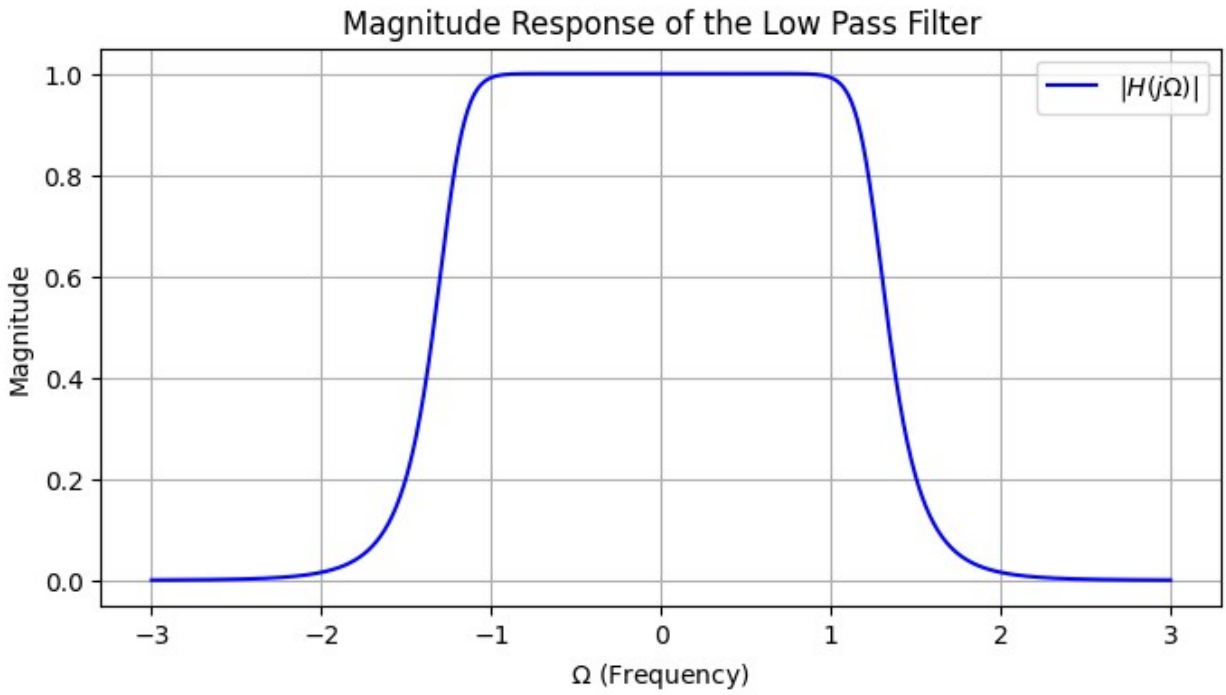


For all poles (p_i) on the left half-plane (stability condition),

$$H_{\text{analog, LPF}}(s) = \frac{(\Omega_C)^N}{\prod_i (s - p_i)}$$

```
from sympy import I, Abs
from numpy.polynomial import Polynomial
from sympy import symbols, Poly
poly = Polynomial([1.0])
for pole in stable_poles:
    poly *= Polynomial([-pole, 1.0])
s, Omega = symbols('s Omega')
N = 9
product_neg_poles = np.product([-pole for pole in stable_poles])
W_C = np.abs(product_neg_poles) ** (1/N)
poly = Polynomial([1.0])
for pole in stable_poles:
    poly *= Polynomial([-pole, 1.0])
numerator = W_C ** N
denominator = Poly(poly.coef[::-1], s).as_expr()
transfer_function = numerator / denominator
print(transfer_function)
H_omega = transfer_function.subs(s, I * Omega)
H_magnitude = Abs(H_omega)
H_func = lambdify(Omega, H_magnitude, modules="numpy")
W_values = np.linspace(-3, 3, 1000)
mag_values = H_func(W_values)
plt.figure(figsize=(8, 4))
plt.plot(W_values, mag_values, label=r'$|H(j\Omega)|$', color='b')
plt.xlabel(r'$\Omega$ (Frequency)')
plt.ylabel('Magnitude')
plt.title('Magnitude Response of the Low Pass Filter')
plt.grid(True)
plt.legend()
plt.show()
```

```
8.0216252828341/(1.0*s**9 + s**8*(7.25777276619235 -
1.55431223447522e-15*I) + s**7*(26.3376327628417 - 1.24344978758018e-
14*I) + s**6*(62.3829842736811 - 4.61852778244065e-14*I) +
s**5*(105.926074560172 - 1.06581410364015e-13*I) +
s**4*(133.498527406636 - 1.98951966012828e-13*I) +
s**3*(124.878288208636 - 2.41584530158434e-13*I) +
s**2*(83.7422685430251 - 1.84741111297626e-13*I) + s*(36.6537610235734
- 9.41469124882133e-14*I) + 8.0216252828341 - 2.37587727269783e-14*I)
```



BANDPASS FILTER TRANSFORMATION

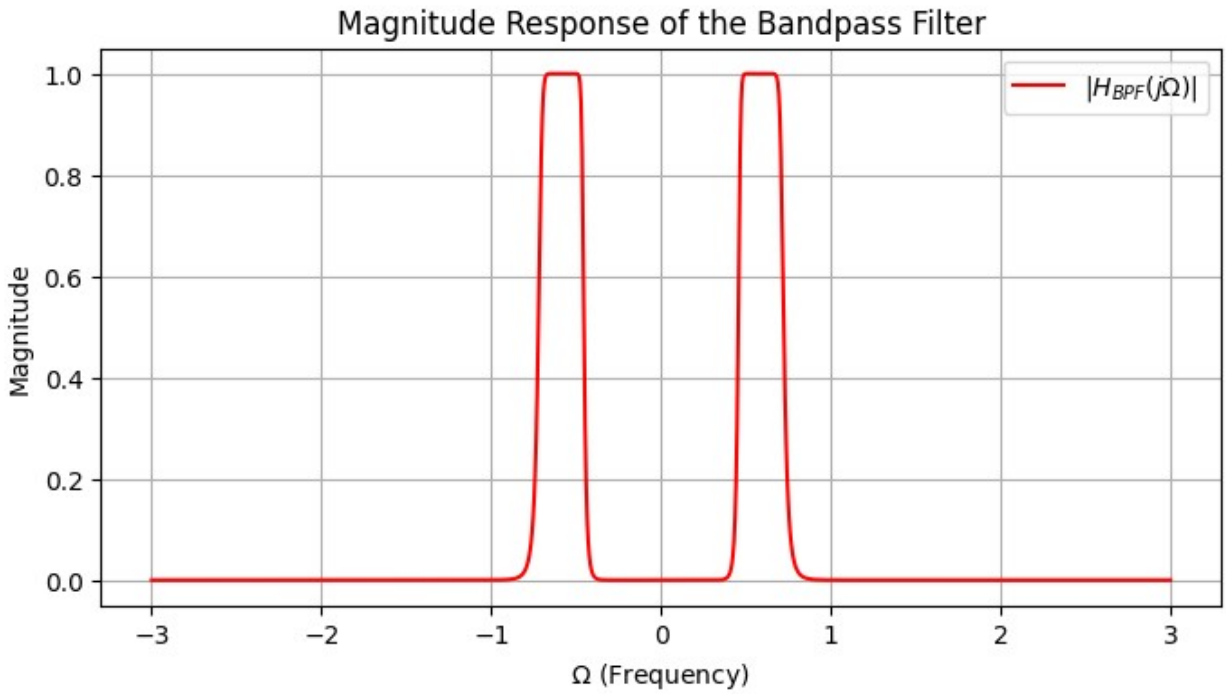
$$\Omega_L = \frac{\Omega^2 - \Omega_0^2}{B\Omega}$$

$$\Omega_0^2 = \Omega_{P1}\Omega_{P2}$$

$$B = \Omega_{P2} - \Omega_{P1}$$

```
H_analog_BPF1 = transfer_function.subs(s, (s**2 + pb1_Omega_sq) /
(pb1_B * s))
H_analog_BPF1 = simplify(H_analog_BPF1)

print("\nBandpass Filter Transfer Function:")
print(H_analog_BPF1)

H_analog_BPF_omega1 = simplify(H_analog_BPF1.subs(s, I * Omega))

print("\nBandpass Filter Transfer Function (Frequency Domain):")
print(H_analog_BPF_omega1)

H_func_BPF = lambdify(Omega, Abs(H_analog_BPF_omega1),
modules="numpy")

W_values = np.linspace(-3, 3, 1000)
mag_values_BPF = H_func_BPF(W_values)
plt.figure(figsize=(8, 4))
```

```
plt.plot(W_values, mag_values_BPF, label=r'$|H_{BPF}(j\Omega)|$',
color='r')
plt.xlabel(r'$\Omega$ (Frequency)')
plt.ylabel('Magnitude')
plt.title('Magnitude Response of the Bandpass Filter')
plt.grid(True)
plt.legend()
plt.show()


Bandpass Filter Transfer Function:
8.0216252828341*s**9/(s**9*(8.0216252828341 - 2.37587727269783e-14*I)
+ 4.99465978484676*s**8*(36.6537610235734 - 9.41469124882133e-
14*I)*(s**2 + 0.328332015709629) +
24.9466263663655*s**7*(83.7422685430251 - 1.84741111297626e-
13*I)*(s**2 + 0.328332015709629)**2 +
124.599911479683*s**6*(124.878288208636 - 2.41584530158434e-
13*I)*(s**2 + 0.328332015709629)**3 +
622.33416706304*s**5*(133.498527406636 - 1.98951966012828e-13*I)*(s**2
+ 0.328332015709629)**4 + 3108.34743696587*s**4*(105.926074560172 -
1.06581410364015e-13*I)*(s**2 + 0.328332015709629)**5 +
15525.1379407449*s**3*(62.3829842736811 - 4.61852778244065e-
14*I)*(s**2 + 0.328332015709629)**6 +
77542.7821268373*s**2*(26.3376327628417 - 1.24344978758018e-
14*I)*(s**2 + 0.328332015709629)**7 +
387299.815494048*s*(7.25777276619235 - 1.55431223447522e-15*I)*(s**2 +
0.328332015709629)**8 + 1934430.81312669*(s**2 +
0.328332015709629)**9)

Bandpass Filter Transfer Function (Frequency Domain):
8.0216252828341*I*Omega**9/(-1934430.81312669*Omega**18 +
Omega**17*(6.01984841632394e-10 + 2810934.05324403*I) +
Omega**16*(7758513.43218801 - 9.64205559639917e-10*I) -
Omega**15*(2.29823998089398e-9 + 8351861.5858318*I) + Omega**14*(-
12530379.3051265 + 2.54734893841823e-9*I) +
Omega**13*(3.35342685416184e-9 + 10475700.6520141*I) +
Omega**12*(10930891.9565338 - 2.75677762165412e-9*I) -
Omega**11*(2.5198805874719e-9 + 7248880.90470978*I) + Omega**10*(-
5733027.85808807 + 1.58173502506627e-9*I) +
Omega**9*(1.08042698904163e-9 + 3027378.36321798*I) +
Omega**8*(1882336.59276551 - 5.19334249098528e-10*I) -
Omega**7*(2.71647946701758e-10 + 781443.225302022*I) + Omega**6*(-
386896.944950909 + 9.75756456077196e-11*I) +
Omega**5*(3.89710197003284e-11 + 121740.760791523*I) +
Omega**4*(47811.2812670869 - 9.71973901303219e-12*I) -
Omega**3*(2.87921948701461e-12 + 10463.1556454874*I) + Omega**2*(-
3191.3258261557 + 3.96608722933407e-13*I) + Omega*(8.13001634929108e-
14 + 379.626499359734*I) + 85.7772853360871)
```



##Band Pass Filter 2

```
pb2_p1 = Omega3[16]
pb2_p2 = Omega3[17]
pb2_s1 = Omega3[15]
pb2_s2 = Omega3[18]
pb2_Omega_sq = pb2_p1*pb2_p2
pb2_B = (pb2_p2-pb2_p1)
pb2_ls1 = (pb2_s1**2 - pb2_Omega_sq)/(pb2_B*pb2_s1)
pb2_ls2 = (pb2_s2**2 - pb2_Omega_sq)/(pb2_B*pb2_s2)
print("Omega_sqared = ",pb2_Omega_sq)
print("B = ",pb2_B)
print('Ω LS1=',pb2_ls1)
print('Ω LS2=',pb2_ls2)
```

```
Omega_sqared =  2.7130806680466875
B =  0.559656862006674
Ω LS1= -1.3131178794147678
Ω LS2= 1.3757905251137534
```

```
LP2_stop = min(abs(pb2_ls1),abs(pb2_ls2))
print("stopband edge for Low pass is ",LP2_stop)
LP2_pass = 1
tol = 0.15
D1 = 1/(1-tol)**2 -1
D2 = 1/tol**2 -1
print("The value of D1 is",D1)
print("The value of D2 is",D2)
```

```
stopband edge for Low pass is  1.3131178794147678
The value of D1 is 0.3840830449826991
The value of D2 is 43.44444444444444
```

```
import numpy as np
N_min = int(np.ceil(np.log(D2/D1)/(2*np.log(LP2_stop/LP2_pass))))
print('N_min = ',N_min)
```

```
N_min =  9
```

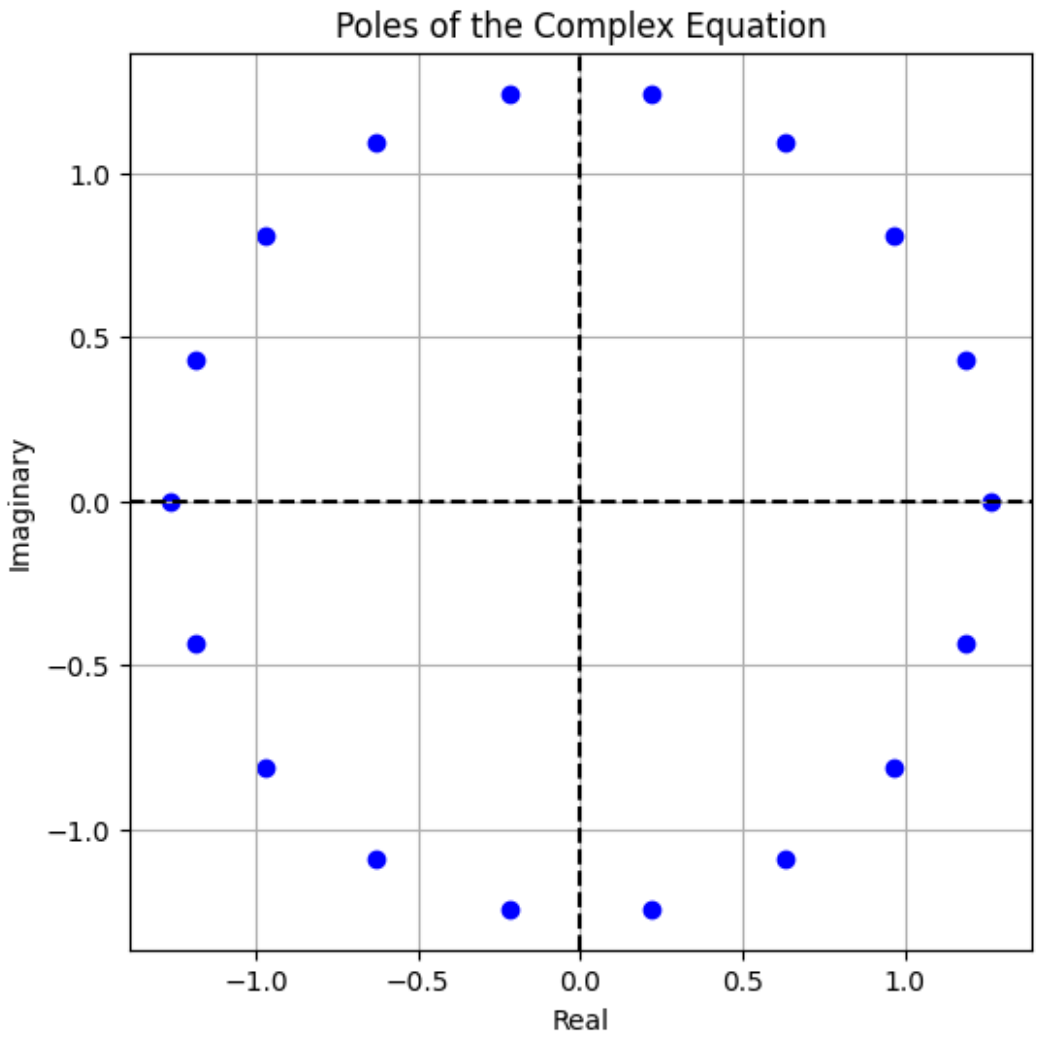
```
N = N_min
leftTerm = LP2_pass/(D1**(1/(2*N)))
rightTerm = LP2_stop/(D2**(1/(2*N)))
print(leftTerm,'<= Ω c <=',rightTerm)
```

```
1.0545993333022403 <= Ω c <= 1.064896421482225
```

```
lp2_omegac = (leftTerm+rightTerm)/2
print("Omega_c = ",lp2_omegac)
```

```
Omega_c =  1.0597478773922326
```

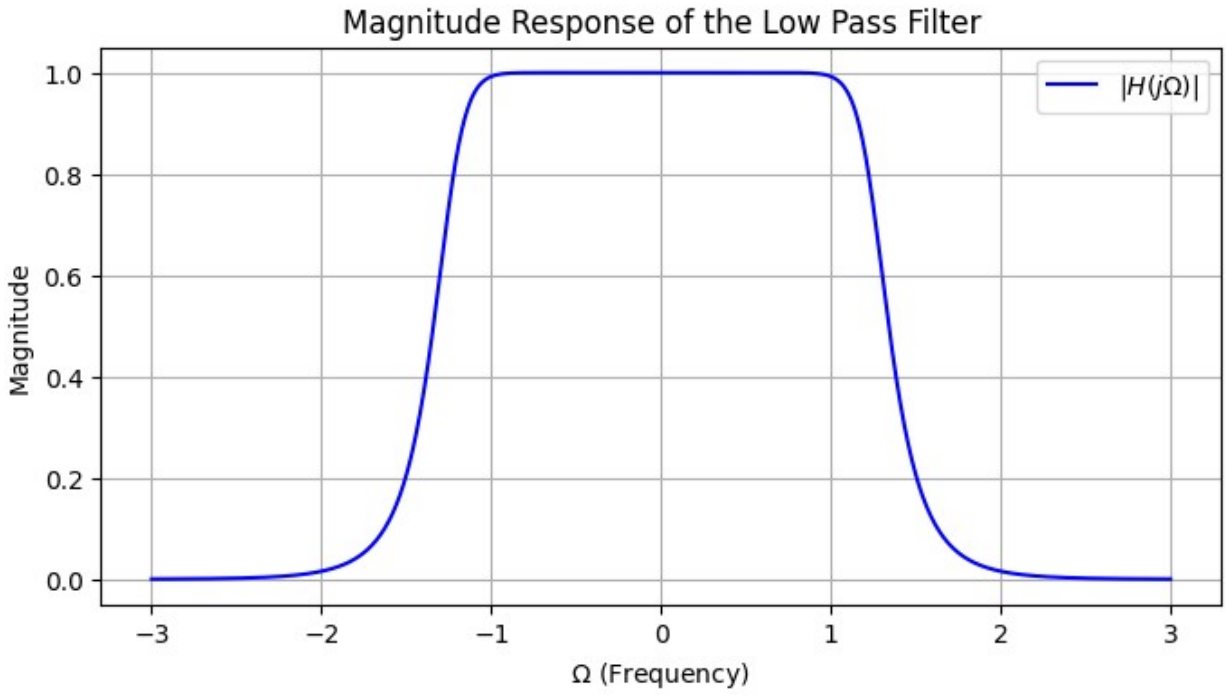
```
poles = [lp2_omegac * np.exp((np.pi + 2 * k * np.pi) / (2 * N)) for k
in range(0, 2*N)*np.exp(1j*np.pi/2)]
plt.figure(figsize=(6, 6))
plt.scatter(np.real(poles), np.imag(poles), color='blue', marker='o')
plt.axhline(y=0, color='k', linestyle='--')
plt.axvline(x=0, color='k', linestyle='--')
plt.ylabel('Imaginary')
plt.xlabel('Real')
plt.title('Poles of the Complex Equation')
plt.grid(True)
plt.show()
stable_poles = [pole for pole in poles if np.real(pole) < 0]
```




```
from numpy.polynomial import Polynomial
from sympy import symbols, Poly
poly = Polynomial([1.0])
for pole in stable_poles:
    poly *= Polynomial([-pole, 1.0])
s, Omega = symbols('s Omega')
N = 9
product_neg_poles = np.product([-pole for pole in stable_poles])
W_C = np.abs(product_neg_poles) ** (1/N)
poly = Polynomial([1.0])
for pole in stable_poles:
    poly *= Polynomial([-pole, 1.0])
numerator = W_C ** N
denominator = Poly(poly.coef[::-1], s).as_expr()
transfer_function = numerator / denominator
print(transfer_function)
H_omega = transfer_function.subs(s, I * Omega)
H_magnitude = Abs(H_omega)
```

```
H_func = lambdify(Omega, H_magnitude, modules="numpy")
W_values = np.linspace(-3, 3, 1000)
mag_values = H_func(W_values)
plt.figure(figsize=(8, 4))
plt.plot(W_values, mag_values, label=r'$|H(j\Omega)|$', color='b')
plt.xlabel(r'$\Omega$ (Frequency)')
plt.ylabel('Magnitude')
plt.title('Magnitude Response of the Low Pass Filter')
plt.grid(True)
plt.legend()
plt.show()
```

```
8.10981925859702/(1.0*s**9 + s**8*(7.26659593708119 -
1.99840144432528e-15*I) + s**7*(26.4017082564024 - 1.59872115546023e-
14*I) + s**6*(62.6107752857165 - 5.6843418860808e-14*I) +
s**5*(106.442105880597 - 1.20792265079217e-13*I) +
s**4*(134.311964053842 - 1.84741111297626e-13*I) +
s**3*(125.791937577666 - 1.98951966012828e-13*I) +
s**2*(84.4575027922567 - 1.70530256582424e-13*I) + s*(37.0117571233157
- 9.05941988094128e-14*I) + 8.10981925859703 - 2.26485497023532e-14*I)
```



$ H_{\text{analog, LPF}}(s) $, $ H_{\text{analog, BPF}}(s)$, and $ H_{\text{analog, BPF}}(\Omega) $

$$\Omega_L = \frac{\Omega^2 - \Omega_0^2}{B\Omega}$$

From analytic continuation

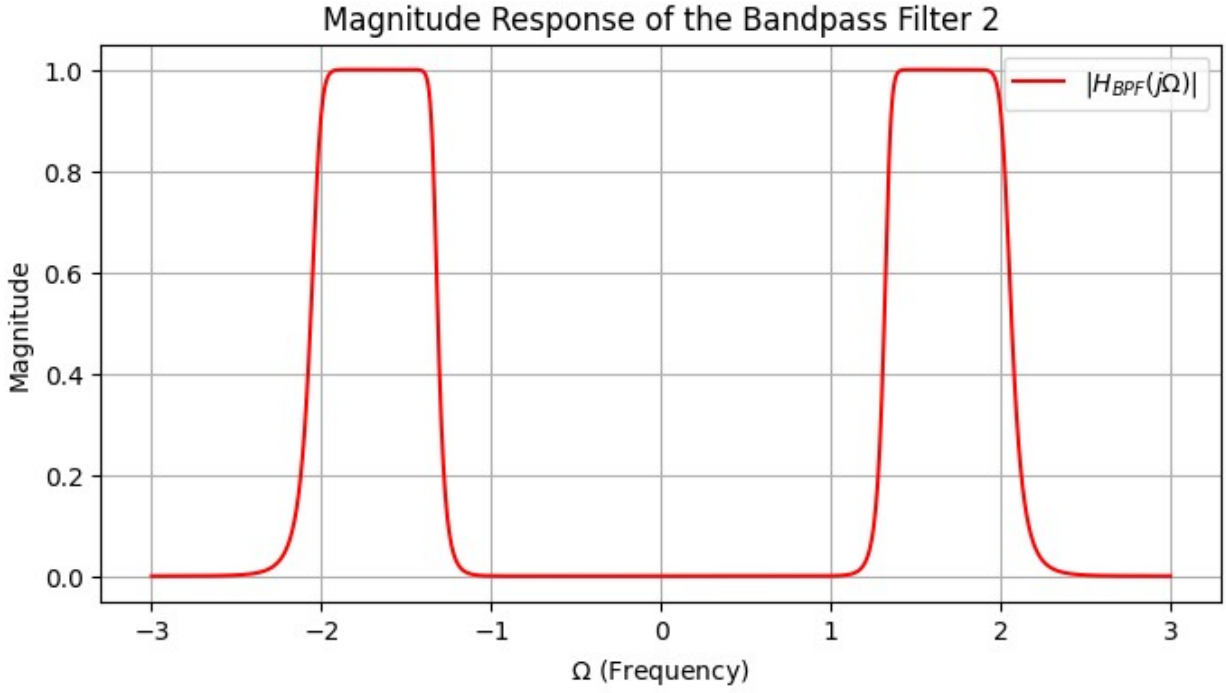
$$\Omega = \frac{s}{j}, \Omega_L = \frac{s_L}{j}$$

$$\frac{s_L}{j} = \frac{(s/j)^2 - \Omega_0^2}{Bs/j}$$

$$\Rightarrow s_L = \frac{s^2 + \Omega_0^2}{Bs}$$

```
H_analog_BPF2 = transfer_function.subs(s, (s**2 + pb2_Omega_sq) /
(pb2_B * s))
H_analog_BPF2 = simplify(H_analog_BPF2)

print("\nBandpass Filter Transfer Function:")
print(H_analog_BPF2)

H_analog_BPF_omega2 = simplify(H_analog_BPF2.subs(s, I * Omega))

print("\nBandpass Filter Transfer Function (Frequency Domain):")
print(H_analog_BPF_omega2)
H_func_BPF = lambdify(Omega, Abs(H_analog_BPF_omega2),
modules="numpy")
W_values = np.linspace(-3, 3, 1000)
mag_values_BPF = H_func_BPF(W_values)
plt.figure(figsize=(8, 4))
plt.plot(W_values, mag_values_BPF, label=r'$|H_{BPF}(j\Omega)|$',
color='r')
plt.xlabel(r'$\Omega$ (Frequency)')
plt.ylabel('Magnitude')
plt.title('Magnitude Response of the Bandpass Filter 2')
plt.grid(True)
plt.legend()
plt.show()


Bandpass Filter Transfer Function:
8.10981925859702*s**9/(185.65365449227*s**18 + s**17*(755.016334001139
- 2.076385897085e-13*I) + s**16*(6068.4903877125 - 9.29650892382003e-
13*I) + s**15*(18424.9577765149 - 6.35662595366787e-12*I) +
s**14*(80291.6615038716 - 1.98555619228829e-11*I) +
s**13*(190148.869950768 - 7.47915957379291e-11*I) +
s**12*(575768.274124227 - 1.74681944338885e-10*I) +
s**11*(1084471.96793542 - 4.57444143348923e-10*I) +
s**10*(2489129.46310055 - 8.21134633248459e-10*I) +
s**9*(3739323.87083656 - 1.61252460166318e-9*I) +
s**8*(6753209.02660353 - 2.22780449933e-9*I) + s**7*(7982588.53982685
- 3.36715792042074e-9*I) + s**6*(11498358.6636 - 3.48847919958532e-
9*I) + s**5*(10302546.3274743 - 4.05231900770827e-9*I) +
```

```
s**4*(11802770.9528072 - 2.91874206020468e-9*I) +
s**3*(7348230.01695789 - 2.53514554583424e-9*I) +
s**2*(6566283.29892913 - 1.00590933469098e-9*I) + s*(2216450.32279287
- 6.0955054673422e-10*I) + 1478658.43144405)

Bandpass Filter Transfer Function (Frequency Domain):
8.10981925859702*I*Omega**9/(-185.65365449227*Omega**18 +
Omega**17*(2.076385897085e-13 + 755.016334001139*I) +
Omega**16*(6068.4903877125 - 9.29650892382003e-13*I) -
Omega**15*(6.35662595366787e-12 + 18424.9577765149*I) + Omega**14*(-
80291.6615038716 + 1.98555619228829e-11*I) +
Omega**13*(7.47915957379291e-11 + 190148.869950768*I) +
Omega**12*(575768.274124227 - 1.74681944338885e-10*I) -
Omega**11*(4.57444143348923e-10 + 1084471.96793542*I) + Omega**10*(-
2489129.46310055 + 8.21134633248459e-10*I) +
Omega**9*(1.61252460166318e-9 + 3739323.87083656*I) +
Omega**8*(6753209.02660353 - 2.22780449933e-9*I) -
Omega**7*(3.36715792042074e-9 + 7982588.53982685*I) + Omega**6*(-
11498358.6636 + 3.48847919958532e-9*I) + Omega**5*(4.05231900770827e-9
+ 10302546.3274743*I) + Omega**4*(11802770.9528072 -
2.91874206020468e-9*I) - Omega**3*(2.53514554583424e-9 +
7348230.01695789*I) + Omega**2*(-6566283.29892913 + 1.00590933469098e-
9*I) + Omega*(6.0955054673422e-10 + 2216450.32279287*I) +
1478658.43144405)
```



#Convert to discrete

```
from sympy import symbols, simplify
z = symbols('z')
H_digital_BPF1 = H_analog_BPF1.subs(s, (z - 1) / (z + 1))
H_digital_BPF1 = simplify(H_digital_BPF1)
print("\nDigital Bandpass Filter Transfer Function:")
print(H_digital_BPF1)
H_digital_BPF2 = H_analog_BPF2.subs(s, (z - 1) / (z + 1))
H_digital_BPF2 = simplify(H_digital_BPF2)
print("\nDigital Bandpass Filter Transfer Function:")
print(H_digital_BPF2)
```

```
Digital Bandpass Filter Transfer Function:
(8.0216252828341*z**18 - 72.1946275455069*z**16 +
288.778510182028*z**14 - 673.816523758065*z**12 +
1010.7247856371*z**10 - 1010.7247856371*z**8 + 673.816523758065*z**6 -
288.778510182028*z**4 + 72.1946275455069*z**2 -
8.0216252828341)/(z**18*(74036347.6144142 - 1.86446321264751e-8*I) +
z**17*(-592950834.810509 + 1.29632993754262e-7*I) +
z**16*(2620868334.49392 - 4.83797263406173e-7*I) + z**15*(-
8026708011.71955 + 1.21616011605916e-6*I) + z**14*(18810736815.9848 -
2.24715756955025e-6*I) + z**13*(-35402987021.6803 + 3.1370265008607e-
6*I) + z**12*(55131022849.9606 - 3.25015334543754e-6*I) + z**11*(-
72296312256.0925 + 2.2073470206513e-6*I) + z**10*(80738028064.7586 -
2.70565260647729e-7*I) - z**9*(77227081202.6563 + 1.74125802365753e-
6*I) + z**8*(63399267095.6968 + 2.98813375088385e-6*I) -
z**7*(44576034010.8698 + 3.13607289590073e-6*I) +
z**6*(26687616179.3623 + 2.46045812825151e-6*I) -
z**5*(13452682063.6287 + 1.51555086224054e-6*I) +
z**4*(5609810152.55637 + 7.39580689764957e-7*I) -
z**3*(1878300465.20524 + 2.8155111720257e-7*I) +
z**2*(481193915.097737 + 8.04550569158167e-8*I) - z*(85416671.1353845
+ 1.57337323240551e-8*I) + 8378782.95990194 + 1.69044535203359e-9*I)

Digital Bandpass Filter Transfer Function:
(8.10981925859702*z**18 - 72.9883733273732*z**16 +
291.953493309493*z**14 - 681.22481772215*z**12 +
1021.83722658322*z**10 - 1021.83722658322*z**8 + 681.22481772215*z**6
- 291.953493309493*z**4 + 72.9883733273732*z**2 -
8.10981925859702)/(z**18*(74133663.8060399 - 2.33730345102045e-8*I) +
z**17*(541644443.257958 - 1.49308731792569e-7*I) +
z**16*(2268936507.42747 - 5.30979129868128e-7*I) +
z**15*(6655743525.60078 - 1.28464363090807e-6*I) +
z**14*(15105739081.5725 - 2.30555837135583e-6*I) +
z**13*(27723265469.8525 - 3.14203763634244e-6*I) +
z**12*(42387162160.1079 - 3.18865665047703e-6*I) +
z**11*(54869177227.6018 - 2.11434396144521e-6*I) +
z**10*(60819668168.6953 - 2.04242231373252e-7*I) +
z**9*(58020391511.5313 + 1.74818043354961e-6*I) +
z**8*(47749266755.6826 + 2.95291466507787e-6*I) +
```

```
z**7*(33817609597.5572 + 3.09907273368195e-6*I) +
z**6*(20506134250.2091 + 2.44990658090581e-6*I) +
z**5*(10525496927.6917 + 1.52875107755062e-6*I) +
z**4*(4499822469.00183 + 7.60385178173692e-7*I) +
z**3*(1555211647.20482 + 2.96530391552816e-7*I) +
z**2*(415821081.507769 + 8.75450326852919e-8*I) + z*(77843580.1354133
+ 1.7799324153289e-8*I) + 8367784.02626957 + 2.05796074177729e-9*I)
```

Final Filter (SUM OF BPF1 and BPF2)

```
H_final_analog_omega = simplify(H_analog_BPF1 + H_analog_BPF2)
H_final_z = simplify(H_digital_BPF1+H_digital_BPF2)

print('H_final_analog(Omega):')
print(H_final_analog_omega)
print()
print('H_final(z):')
print(H_final_z)
print()
```

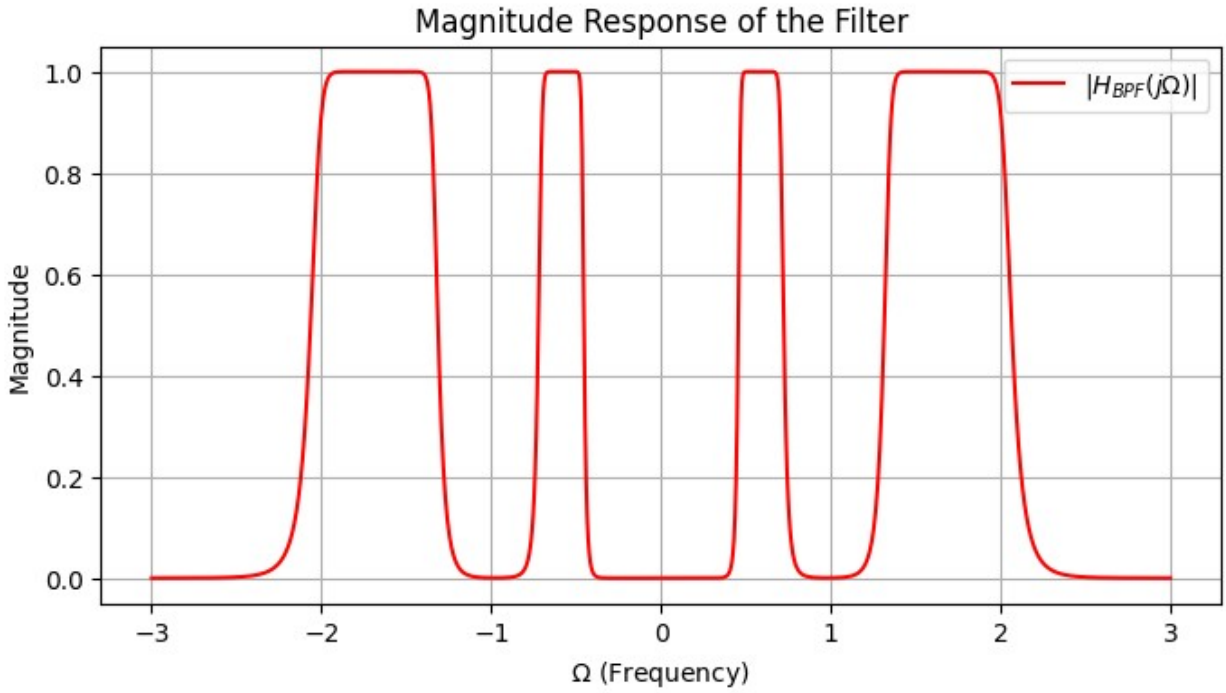
```
H_final_analog(Omega):
s**9*(15689373.5067671*s**18 + s**17*(22802223.5777584 -
4.88365386101477e-9*I) + s**16*(62968820.8063647 - 7.82699012791666e-
9*I) + s**15*(67879886.0410507 - 1.86893013293951e-8*I) +
s**14*(102263181.02816 - 2.08178133566763e-8*I) +
s**13*(86481341.8777008 - 2.7795635839493e-8*I) +
s**12*(93266155.4475096 - 2.37582013489233e-8*I) +
s**11*(67486341.7208072 - 2.41052216234165e-8*I) +
s**10*(66460683.5670505 - 1.94144195029564e-8*I) +
s**9*(54546946.2560927 - 2.16971357171311e-8*I) +
s**8*(69437121.819238 - 2.20823197920254e-8*I) +
s**7*(70370697.3715913 - 2.92130948554624e-8*I) +
s**6*(95373188.8622835 - 2.8774593795954e-8*I) +
s**5*(83630461.6644609 - 3.28222325324348e-8*I) +
s**4*(95065146.7321361 - 2.34918804308456e-8*I) +
s**3*(59029601.9892693 - 2.03593375557737e-8*I) +
s**2*(52698145.2005866 - 8.07224417645539e-9*I) + s*(17782612.6497567
- 4.89024540648019e-9*I) + 11861939.496628)/(359134149.819424*s**36 +
s**35*(1982387040.42728 - 5.13423171724298e-7*I) +
s**34*(15301772291.2666 - 3.01552038833612e-6*I) +
s**33*(60108290330.1463 - 2.13284140738261e-5*I) +
s**32*(262824572106.779 - 8.43750956737301e-5*I) +
s**31*(798563365932.488 - 0.00034276843932159*I) +
s**30*(2511085627270.41 - 0.0010218393598187*I) +
s**29*(6166182225800.25 - 0.00299798548071203*I) +
s**28*(15190601556595.0 - 0.00711149532027537*I) +
s**27*(30926899783901.1 - 0.0163464931479212*I) +
s**26*(62199095893315.2 - 0.0318009738319192*I) +
s**25*(106631561004736.0 - 0.0595549384915286*I) +
```

```
s**24*(179057033342124.0 - 0.0967376842633099*I) +
s**23*(260916962474262.0 - 0.150801954842073*I) +
s**22*(370577519142347.0 - 0.206729403124789*I) +
s**21*(461432553504748.0 - 0.271588690613759*I) +
s**20*(558420665720927.0 - 0.316124660869924*I) +
s**19*(595638352976106.0 - 0.352424451341025*I) +
s**18*(616597872893825.0 - 0.349244985172581*I) +
s**17*(563570484990676.0 - 0.331446963910293*I) +
s**16*(499781752065582.0 - 0.279574398291826*I) +
s**15*(390774631436945.0 - 0.225900601920538*I) +
s**14*(296726136245931.0 - 0.161649657316346*I) +
s**13*(197672304683035.0 - 0.110890014072603*I) +
s**12*(128178935955467.0 - 0.0668324684937628*I) +
s**11*(72210383612052.5 - 0.0386757357935391*I) +
s**10*(39766558552877.9 - 0.019382945955433*I) +
s**9*(18700874293260.4 - 0.00935789917902746*I) +
s**8*(8662824454014.01 - 0.00381537598918311*I) +
s**7*(3325160906972.05 - 0.00150885403440578*I) +
s**6*(1275312775726.45 - 0.000481135628634345*I) +
s**5*(383503341666.663 - 0.000151162338475943*I) +
s**4*(118644760834.924 - 3.47562554221058e-5*I) +
s**3*(25667894837.6879 - 8.21488775255989e-6*I) +
s**2*(6123542073.50672 - 1.08433231803026e-6*I) + s*(751459015.849328
- 1.72500763400562e-7*I) + 126835306.188588)

H_final(z):
(z**36*(1195093869.61521 - 3.38694321252355e-7*I) -
z**35*(463855339.202812 + 1.46398548597265e-7*I) +
z**34*(28699482119.9144 - 5.13457508556416e-6*I) + z**33*(-
7530572622.73433 + 8.75495839005602e-7*I) + z**32*(-38350308980.6033 +
2.4734053169075e-5*I) + z**31*(23922465124.1425 - 1.05505847606493e-
6*I) + z**30*(-356397840551.357 + 1.23975983353845e-5*I) +
z**29*(53939666711.4586 - 4.80531555137074e-6*I) +
z**28*(749001856620.26 - 0.000213584789497346*I) + z**27*(-
250694886286.696 + 1.86379000152538e-5*I) + z**26*(777540960193.628 +
0.000308687065868108*I) + z**25*(116146111948.14 - 2.2946144211674e-
5*I) + z**24*(-3058081663454.08 + 0.000209075175662407*I) +
z**23*(554426952802.006 - 4.2152697908044e-6*I) +
z**22*(928480730009.748 - 0.000889917552680006*I) + z**21*(-
743647670898.797 + 4.48078544527941e-5*I) + z**20*(4002381897700.47 +
0.000561250597429262*I) - z**19*(157052661876.117 + 5.00997656472622e-
5*I) + z**18*(-3435738540031.74 + 0.000503207561905928*I) +
z**17*(822075913891.0 + 1.24091837208696e-5*I) -
z**16*(1572974131302.34 + 0.000789794748652671*I) + z**15*(-
332370982036.781 + 2.1755220312543e-5*I) + z**14*(2719372818288.11 +
0.000153770655262 17*I) - z**13*(272186834998.514 + 2.21693061719771e-
5*I) + z**12*(-184045597192.402 + 0.000263107764272888*I) +
z**11*(213357242232.04 + 6.38734232690955e-6*I) -
z**10*(822633786194.084 + 0.000141411228662763*I) +
```

```
z**9*(13803461925.799 + 2.03205627301661e-6*I) +
z**8*(184848468957.555 - 1.69816557444703e-5*I) -
z**7*(38244649204.9995 + 1.83309599474595e-6*I) +
z**6*(104169405415.441 + 2.30384151823941e-5*I) +
z**5*(2309200284.94588 + 3.39174199670894e-7*I) -
z**4*(21311478034.3041 + 9.92652195450695e-7*I) +
z**3*(2142816610.25098 + 4.13090356675045e-8*I) -
z**2*(6022293790.47334 + 1.08277285190265e-6*I) + z*(68281734.0584358
- 1.51817832341505e-8*I) - 135073643.318237 -
3.02173961888485e-8*I)/(z**36*(5.48858570347408e+15 -
3.11264899765259*I) + z**35*(-3.85624155689099e+15 +
2.31619454527352*I) + z**34*(4.11098191679231e+16 - 19.990823747902*I)
+ z**33*(-2.80713582380901e+16 + 14.1716403704303*I) +
z**32*(1.65315961978281e+17 - 66.2033196887443*I) + z**31*(-
1.08516740635357e+17 + 44.5289059570306*I) +
z**30*(4.57701626795491e+17 - 145.380336707399*I) + z**29*(-
2.87386299944567e+17 + 91.4875809679302*I) +
z**28*(9.64936769825604e+17 - 229.876774169757*I) + z**27*(-
5.76790383187561e+17 + 132.947738572828*I) +
z**26*(1.62975805227112e+18 - 265.211534396745*I) + z**25*(-
9.23353616717677e+17 + 135.883960505089*I) +
z**24*(2.27251108268042e+18 - 207.35519367081*I) + z**23*(-
1.21441594349414e+18 + 83.8403660519689*I) +
z**22*(2.66387944104709e+18 - 64.0995259705232*I) -
z**21*(1.33549580688162e+18 + 4.71420032804599*I) +
z**20*(2.65427981394588e+18 + 102.294414447068*I) -
z**19*(1.24033251502659e+18 + 87.0584209645167*I) +
z**18*(2.26094669995503e+18 + 217.564383892386*I) -
z**17*(9.77156045911392e+17 + 127.845631336473*I) +
z**16*(1.64916391009589e+18 + 245.329007140605*I) -
z**15*(6.52771550743871e+17 + 121.07424836618*I) +
z**14*(1.02763513405373e+18 + 201.417641847736*I) -
z**13*(3.67877341674881e+17 + 86.1821794549138*I) +
z**12*(5.43524050011111e+17 + 129.442035561637*I) -
z**11*(1.73031950334214e+17 + 48.0141402979298*I) +
z**10*(2.41152604903416e+17 + 66.5051706525192*I) -
z**9*(6.66949417675653e+16 + 21.0425068435959*I) +
z**8*(8.80830138511974e+16 + 27.3290174000825*I) -
z**7*(2.04386300334449e+16 + 7.12941362036966*I) +
z**6*(2.56845680841042e+16 + 8.8173504121159*I) -
z**5*(4.72898314173696e+15 + 1.78868194139858*I) +
z**4*(5.68065520491862e+15 + 2.14492420805831*I) -
z**3*(746627260363392.0 + 0.300250179296678*I) +
z**2*(861461863970702.0 + 0.35482329851569*I) - z*(62513793527429.3 +
0.0267136377658394*I) + 70111846211447.2 + 0.031388488009379*I)


H_analog_final_omega = simplify(H_final_analog_omega.subs(s, I *
Omega))
```

Evidence Of Correctness

```
H_func_final = lambdify(Omega, Abs(H_analog_final_omega),
modules="numpy")
W_values = np.linspace(-3, 3, 1000)
mag_values_final = H_func_final(W_values)
plt.figure(figsize=(8, 4))
plt.plot(W_values, mag_values_final, label=r'$|H_{BPF}(j\Omega)|$',
color='r')
plt.xlabel(r'$\Omega$ (Frequency)')
plt.ylabel('Magnitude')
plt.title('Magnitude Response of the Filter')
plt.grid(True)
plt.legend()
plt.show()
```



The magnitude responses match the target filter magnitude responses as defined at the start of the document. Hence, we can say that the filter design was successful.